CENG 3420 Lab2 Report

Name: Tam Rocky Lok Ki

SID: 1155158247

Lab2.1

Integer Register-Immediate Instructions:

```
if (is_opcode(opcode) == ADDI) {
    binary = (0x04 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);

} else if (is_opcode(opcode) == SLLI) {
    /* Lab2-1 assignment */
    binary = (0x04 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x01 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);

} else if (is_opcode(opcode) == XORI) {
    /* Lab2-1 assignment */
    binary = (0x04 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x04 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);


} else if (is_opcode(opcode) == SRLI) {
    /* Lab2-1 assignment */
    binary = (0x04 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x05 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);

} else if (is_opcode(opcode) == SRAI) {
    /* Lab2-1 assignment */
    binary = (0x04 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x05 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);
    binary += 0x01 << 30;
```

Figure 1.1 Integer Register-Immediate Instructions (ADDI, SLLI, ARLI, SRAI)

```
} else if (is_opcode(opcode) == ORI) {
    /* Lab2-1 assignment */
    binary = (0x04 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x06 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);

} else if (is_opcode(opcode) == ANDI) {
    /* Lab2-1 assignment */
    binary = (0x04 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x03 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (MASK11_0(validate_imm(arg3, 12, line_no)) << 20);

} else if (is_opcode(opcode) == LUI) {
     /* Lab2-1 assignment */
     binary = (0x0D << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += handle_label_or_imm(arg2, label_table, cmd_no, line_no) & 0xFFFFF000;
}
```

Figure 1.2 Integer Register-Immediate Instructions (ORI, ANDI, LUI)

Integer Register-Register Operations:

```
 else if (is_opcode(opcode) == ADD) {
     binary = (0x0C << 2) + 0x03;
     binary += (reg_to_num(arg1, line_no) << 7);
     binary += (reg_to_num(arg2, line_no) << 15);
     binary += (reg_to_num(arg3, line_no) << 20);

 } else if (is_opcode(opcode) == SUB) {
     /* Lab2-1 assignment */
     binary = (0x0C << 2) + 0x03;
     binary += (reg_to_num(arg1, line_no) << 7);
     binary += (reg_to_num(arg2, line_no) << 15);
     binary += (reg_to_num(arg3, line_no) << 20);
     binary += 0x01 << 30;

 } else if (is_opcode(opcode) == SLL) {
     /* Lab2-1 assignment */
     binary = (0x0C << 2) + 0x03;
     binary += (reg_to_num(arg1, line_no) << 7);
     binary += 0x01 << 12;
     binary += (reg_to_num(arg2, line_no) << 15);
     binary += (reg_to_num(arg3, line_no) << 20);

 } else if (is_opcode(opcode) == XOR) {
     /* Lab2-1 assignment */
     binary = (0x0C << 2) + 0x03;
     binary += (reg_to_num(arg1, line_no) << 7);
     binary += 0x04 << 12;
     binary += (reg_to_num(arg2, line_no) << 15);
     binary += (reg_to_num(arg3, line_no) << 20);
```

```
} else if (is_opcode(opcode) == SRL) {
    /* Lab2-1 assignment */
    binary = (0x0C << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x05 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (reg_to_num(arg3, line_no) << 20);

} else if (is_opcode(opcode) == SRA) {
    /* Lab2-1 assignment */
    binary = (0x0C << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x05 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (reg_to_num(arg3, line_no) << 20);
    binary += 0x01 << 30;

} else if (is_opcode(opcode) == OR) {
    /* Lab2-1 assignment */
    binary = (0x0C << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x06 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (reg_to_num(arg3, line_no) << 20);

} else if (is_opcode(opcode) == AND) {
    /* Lab2-1 assignment */
    binary = (0x0C << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x07 << 12;
    binary += (reg_to_num(arg2, line_no) << 15);
    binary += (reg_to_num(arg3, line_no) << 20);
```

Figure 2 Integer Register-Register Operations (ADD, SUB, SLL, XOR, SRL, SRA, OR, AND)

Unconditional Jumps:

```
else if (is_opcode(opcode) == JALR) {
    /*
     * Lab2-1 assignment
     * tip: you may need the function `parse_regs_indirect_addr`
     * e.g., parse_regs_indirect_addr(arg2, line_no)
     */
    binary = (0x19 << 2) + 0x03;
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += reg_to_num(ret->reg, line_no) << 15;
    binary += (ret->imm <<20);

} else if (is_opcode(opcode) == JAL) {
    /*
     * Lab2-1 assignment
     * tip: you may need the function `handle_label_or_imm`
     * e.g., handle_label_or_imm(arg2, label_table, cmd_no, line_no)
     */
    binary = (0x1b << 2) + 0x03;
    binary += reg_to_num(arg1, line_no) << 7;
    int offset;
    offset = handle_label_or_imm(arg2, label_table, cmd_no, line_no);
    // imm[19:12]
    binary += ((offset & 0xFF000) << 11);
    // imm[11]
    binary += ((offset & 0x800) << 8);
    // imm[10:1]
    binary += ((offset & 0x7FE) << 20);
    // imm[20]
    binary += ((offset & 0x1000) << 10);
}
```

Figure 3 Unconditional Jumps (JALR, JAL)

Conditional Branches:

```
else if (is_opcode(opcode) == BEQ) {
    binary = (0x18 << 2) + 0x03;
    binary += (reg_to_num(arg1, line_no) << 15);
    binary += (reg_to_num(arg2, line_no) << 20);
    int offset;
    offset = label_to_num(arg3, 12, label_table, cmd_no, line_no);
    // imm[11]
    binary += ((offset & 0x800) >> 4);
    // imm[4:1]
    binary += ((offset & 0x1E) << 7);
    // imm[10:5]
    binary += ((offset & 0x7E0) << 20);
    // imm[12]
    binary += ((offset & 0x1000) << 19);

} else if (is_opcode(opcode) == BNE) {
    /* Lab2-1 assignment */
    binary = (0x18 << 2) + 0x03;
    binary += 0x01 << 12;
    binary += (reg_to_num(arg1, line_no) << 15);
    binary += (reg_to_num(arg2, line_no) << 20);
    int offset;
    offset = label_to_num(arg3, 12, label_table, cmd_no, line_no);
    // imm[11]
    binary += ((offset & 0x800) >> 4);
    // imm[4:1]
    binary += ((offset & 0x1E) << 7);
    // imm[10:5]
    binary += ((offset & 0x7E0) << 20);
    // imm[12]
    binary += ((offset & 0x1000) << 19);
```

Figure 4.1 Conditional Branches (BEQ, BNE)

```
} else if (is_opcode(opcode) == BLT) {
    /* Lab2-1 assignment */
    binary = (0x18 << 2) + 0x03;
    binary += 0x04 << 12;
    binary += (reg_to_num(arg1, line_no) << 15);
    binary += (reg_to_num(arg2, line_no) << 20);
    int offset;
    offset = label_to_num(arg3, 12, label_table, cmd_no, line_no);
    // imm[11]
    binary += ((offset & 0x800) >> 4);
    // imm[4:1]
    binary += ((offset & 0x1E) << 7);
    // imm[10:5]
    binary += ((offset & 0x7E0) << 20);
    // imm[12]
    binary += ((offset & 0x1000) << 19);

} else if (is_opcode(opcode) == BGE) {
    /* Lab2-1 assignment */
    binary = (0x18 << 2) + 0x03;
    binary += 0x05 << 12;
    binary += (reg_to_num(arg1, line_no) << 15);
    binary += (reg_to_num(arg2, line_no) << 20);
    int offset;
    offset = label_to_num(arg3, 12, label_table, cmd_no, line_no);
    // imm[11]
    binary += ((offset & 0x800) >> 4);
    // imm[4:1]
    binary += ((offset & 0x1E) << 7);
    // imm[10:5]
    binary += ((offset & 0x7E0) << 20);
    // imm[12]
    binary += ((offset & 0x1000) << 19);

}
```

Figure 4.2 Conditional Branches (BLT, BGE)

Load and Store Instructions:

```
else if (is_opcode(opcode) == LB) {
    binary = 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += (reg_to_num(ret->reg, line_no) << 15);
    binary += (ret->imm << 20);
} else if (is_opcode(opcode) == LH) {
    /* Lab2-1 assignment */
    binary = 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x01 << 12;
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += (reg_to_num(ret->reg, line_no) << 15);
    binary += (ret->imm << 20);

} else if (is_opcode(opcode) == LW) {
    /* Lab2-1 assignment */
    binary = 0x03;
    binary += (reg_to_num(arg1, line_no) << 7);
    binary += 0x02 << 12;
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += (reg_to_num(ret->reg, line_no) << 15);
    binary += (ret->imm << 20);

} else if (is_opcode(opcode) == SB) {
    /* Lab2-1 assignment */
    binary += (0x08 << 2) + 0x03;
    binary += reg_to_num(arg1, line_no) << 20;
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += reg_to_num(ret->reg, line_no) << 15;
    int offset;
    offset = ret->imm;
    binary += ((offset & 0x1f) << 7);
    binary += ((offset & 0xfE0) << 20);
```

Figure 5.1 Load (LB, LH, LW)

```
} else if (is_opcode(opcode) == SB) {
    /* Lab2-1 assignment */
    binary += (0x08 << 2) + 0x03;
    binary += reg_to_num(arg1, line_no) << 20;
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += reg_to_num(ret->reg, line_no) << 15;
    int offset;
    offset = ret->imm;
    binary += ((offset & 0x1f) << 7);
    binary += ((offset & 0xfE0) << 20);

} else if (is_opcode(opcode) == SH) {
    /* Lab2-1 assignment */
    binary += (0x08 << 2) + 0x03;
    binary += 0x01 << 12;
    binary += reg_to_num(arg1, line_no) << 20;
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += reg_to_num(ret->reg, line_no) << 15;
    int offset;
    offset = ret->imm;
    binary += ((offset & 0x1f) << 7);
    binary += ((offset & 0xfE0) << 20);

} else if (is_opcode(opcode) == SW) {
    /* Lab2-1 assignment */
    binary += (0x08 << 2) + 0x03;
    binary += 0x02 << 12;
    binary += reg_to_num(arg1, line_no) << 20;
    struct_regs_indirect_addr* ret = parse_regs_indirect_addr(arg2, line_no);
    binary += reg_to_num(ret->reg, line_no) << 15;
    int offset;
    offset = ret->imm;
    binary += ((offset & 0x1f) << 7);
    binary += ((offset & 0xfE0) << 20);

}
```

Figure 5.2 Store (SB, SH, SW)

Console Result:

```
bash tools/validate.sh
tools/../benchmarks/isa.asm
Processing input file tools/../benchmarks/isa.asm
Writing result to output file isa.bin
tools/../benchmarks/swap.asm
Processing input file tools/../benchmarks/swap.asm
Writing result to output file swap.bin
tools/../benchmarks/count10.asm
Processing input file tools/../benchmarks/count10.asm
Writing result to output file count10.bin
[INFO]: You have passed the Lab.
```

Lab2.2

Integer Register-Immediate Instructions:

```
void handle_addi(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = sext(MASK31_20(cur_inst), 12);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] + imm12;
}



void handle_slli(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int shamt = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd]= (signed int) CURRENT_LATCHES.REGS[rs1] << (shamt & 31);

}



void handle_xori(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = sext(MASK31_20(cur_inst), 12);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] ^ imm12;
}


void handle_srli(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int shamt = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = (signed int)(unsigned int)CURRENT_LATCHES.REGS[rs1] >> (shamt & 31);
}
```

Figure 1.1 Integer Register-Immediate Instructions (ADDI, SLLI, XORI, SRLI)

```c
void handle_srai(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    signed int shamt = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] =(signed int) CURRENT_LATCHES.REGS[rs1] >> (shamt & 31);
}


void handle_ori(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = sext(MASK31_20(cur_inst), 12);

    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] | imm12;
}


void handle_andi(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = sext(MASK31_20(cur_inst), 12);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] & imm12;
}


void handle_lui(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst);
    unsigned int imm20 = MASK31_12(cur_inst);
    NEXT_LATCHES.REGS[rd] = imm20 << 12;
}
```

Figure 1.2 Integer Register-Immediate Instructions (SRAI, ORI, ANDI, LUI)

Integer Register-Register Operations:

```
void handle_add(unsigned int cur_inst) {
    unsigned int rd = MASK11_7(cur_inst),
        rs1 = MASK19_15(cur_inst),
        rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] + CURRENT_LATCHES.REGS[rs2];
}


void handle_sub(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst),
        rs1 = MASK19_15(cur_inst),
        rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] - CURRENT_LATCHES.REGS[rs2];
}


void handle_sll(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst),
        rs1 = MASK19_15(cur_inst),
        rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] << NEXT_LATCHES.REGS[rs2];
}


void handle_xor(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst),
        rs1 = MASK19_15(cur_inst),
        rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] ^ CURRENT_LATCHES.REGS[rs2];
}
```

Figure 2.1 Integer Register-Register Operations (ADD, SUB, SLL, XOR)

```
void handle_srl(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst),
        rs1 = MASK19_15(cur_inst),
        rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd]= (signed int)(unsigned int)CURRENT_LATCHES.REGS[rs1]>>(NEXT_LATCHES.REGS[rs2]& 31) ;
}

void handle_sra(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst),
        rs1 = MASK19_15(cur_inst),
        rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd]= (signed int)CURRENT_LATCHES.REGS[rs1] >> (CURRENT_LATCHES.REGS[rs2] & 31);
}


void handle_or(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst),
        rs1 = MASK19_15(cur_inst),
        rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] | CURRENT_LATCHES.REGS[rs2];
}

void handle_and(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst),
            rs1 = MASK19_15(cur_inst),
            rs2 = MASK24_20(cur_inst);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.REGS[rs1] & CURRENT_LATCHES.REGS[rs2];
}
```

Figure 2.2 Integer Register-Register Operations (SRL, SRA, OR, AND)

Unconditional Jumps:

```c
void handle_jalr(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rs1 = MASK19_15(cur_inst);
    unsigned int rd = MASK11_7(cur_inst);
    int imm12 = sext((MASK31_20(cur_inst)),12);

    NEXT_LATCHES.PC =  CURRENT_LATCHES.REGS[rs1] + imm12;
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.PC + 4;
}


void handle_jal(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst);
    int imm20 = (MASK31(cur_inst) << 20) + \
        (MASK19_12(cur_inst) << 12) + \
        (MASK20(cur_inst) << 11) + \
        (MASK30_21(cur_inst) << 1);
    /*
     * the offset is sign-extended and added to the address of the
     * jump instruction to form the jump target address.
     * we directly assign the immediate value to `PC`, since we already
     * encode the jump target address into the immediate value
     */
    NEXT_LATCHES.PC = sext(imm20, 20);
    NEXT_LATCHES.REGS[rd] = CURRENT_LATCHES.PC + 4;
}
```

Figure 3 Unconditional Jumps (JALR, JAL)

Conditional Branches:

```
void handle_beq(unsigned int cur_inst) {
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = (MASK31(cur_inst) << 12) + \
            (MASK7(cur_inst) << 11) + \
            (MASK30_25(cur_inst) << 5) + \
            (MASK11_8(cur_inst) << 1);
    if (CURRENT_LATCHES.REGS[rs1] == CURRENT_LATCHES.REGS[rs2])
        NEXT_LATCHES.PC = sext(imm12, 12);
}


void handle_bne(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = (MASK31(cur_instr) << 12) + \
            (MASK7(cur_inst) << 11) + \
            (MASK30_25(cur_inst) << 5) + \
            (MASK11_8(cur_inst) << 1);
    if (CURRENT_LATCHES.REGS[rs1] != CURRENT_LATCHES.REGS[rs2])
        NEXT_LATCHES.PC = sext(imm12, 12);
}
```

Figure 4.1 Conditional Branches (BEQ, BNE)

```
void handle_blt(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = (MASK31(cur_inst) << 12) + \
            (MASK7(cur_inst) << 11) + \
            (MASK30_25(cur_inst) << 5) + \
            (MASK11_8(cur_inst) << 1);
    if (CURRENT_LATCHES.REGS[rs1] < CURRENT_LATCHES.REGS[rs2])
        NEXT_LATCHES.PC = sext(imm12, 12);
}


void handle_bge(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = (MASK31(cur_inst) << 12) + \
            (MASK7(cur_inst) << 11) + \
            (MASK30_25(cur_inst) << 5) + \
            (MASK11_8(cur_inst) << 1);
    if (CURRENT_LATCHES.REGS[rs1] >= CURRENT_LATCHES.REGS[rs2])
        NEXT_LATCHES.PC = sext(imm12, 12);
}
```

Figure 4.2 Conditional Branches (BLT, BGE)

Load Instructions:

```
void handle_lb(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    signed int imm12 = sext((MASK31_20(cur_inst)),12);
    NEXT_LATCHES.REGS[rd] = sext(MEMORY[CURRENT_LATCHES.REGS[rs1]+imm12],8);
}


void handle_lh(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = sext((MASK31_20(cur_inst)),12);
    NEXT_LATCHES.REGS[rd] = sext(((MEMORY[CURRENT_LATCHES.REGS[rs1]+imm12+1]<<8) +
                                  (MEMORY[CURRENT_LATCHES.REGS[rs1]+imm12])),16);
}

void handle_lw(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rd = MASK11_7(cur_inst), rs1 = MASK19_15(cur_inst);
    int imm12 = sext((MASK31_20(cur_inst)),12);
    int address = CURRENT_LATCHES.REGS[rs1];
    NEXT_LATCHES.REGS[rd] = ((MEMORY[CURRENT_LATCHES.REGS[rs1]+imm12+3]<<24) +
                             (MEMORY[CURRENT_LATCHES.REGS[rs1]+imm12+2]<<16) +
                             (MEMORY[CURRENT_LATCHES.REGS[rs1]+imm12+1]<<8) +
                             (MEMORY[CURRENT_LATCHES.REGS[rs1]+imm12]));
}
```

Figure 5 Load Instructions (LB, LH, LW)

Store Instructions:

```c
void handle_sb(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rs1 = MASK19_15(cur_inst),rs2 = MASK24_20(cur_inst);
    int imm12 = sext((MASK31_25(cur_inst)<<5) + (MASK11_7(cur_inst)) ,12);
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12] = sext(CURRENT_LATCHES.REGS[rs2],8);
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+1] = sext(CURRENT_LATCHES.REGS[rs2],8)>>8;
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+2] = sext(CURRENT_LATCHES.REGS[rs2],8)>>16;
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+3] = sext(CURRENT_LATCHES.REGS[rs2],8)>>24;
}


void handle_sh(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rs1 = MASK19_15(cur_inst),rs2 = MASK24_20(cur_inst);
    int imm12 = sext((MASK31_25(cur_inst)<<5) + (MASK11_7(cur_inst)) ,12);
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12] = sext(CURRENT_LATCHES.REGS[rs2],16);
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+1] = sext(CURRENT_LATCHES.REGS[rs2],16)>>8;
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+2] = sext(CURRENT_LATCHES.REGS[rs2],16)>>16
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+3] = sext(CURRENT_LATCHES.REGS[rs2],16)>>24
}


void handle_sw(unsigned int cur_inst) {
    /*
     * Lab2-2 assignment
     */
    unsigned int rs1 = MASK19_15(cur_inst), rs2 = MASK24_20(cur_inst);
    int imm12 = sext((MASK31_25(cur_inst)<<5) + (MASK11_7(cur_inst)) ,12);
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12] = CURRENT_LATCHES.REGS[rs2];
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+1] = CURRENT_LATCHES.REGS[rs2]>>8;
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+2] = CURRENT_LATCHES.REGS[rs2]>>16;
    MEMORY[NEXT_LATCHES.REGS[rs1]+imm12+3] = CURRENT_LATCHES.REGS[rs2]>>24;
    NEXT_LATCHES.REGS[rs1]=CURRENT_LATCHES.REGS[rs2]<<imm12;
}
```

Figure 6 Store Instructions (SB, SH, SW)

Console Result:

```
bash tools/validate.sh
tools/../benchmarks/isa.asm
Processing input file tools/../benchmarks/isa.asm
Writing result to output file isa.bin
tools/../benchmarks/swap.asm
Processing input file tools/../benchmarks/swap.asm
Writing result to output file swap.bin
tools/../benchmarks/count10.asm
Processing input file tools/../benchmarks/count10.asm
Writing result to output file count10.bin
[INFO]: You have passed the Lab.
```